AGENDA

1. Icebreaker
2. Overview
3. Containment/Program Structure: Knowing the logical containment and inheritance relationships between various objects.
4. Enums.
5. Inheritance & Interfaces.
6. Keeping track of the Canvas both logically and graphically: Layer class, Command Stacks.
7. Examples and Other Concepts: Parsing, Demos.
ICEBREAKER!

If I had a gun with two bullets and I was in a room with Hitler, Bin Laden and Toby, I would shoot Toby twice.
OVERVIEW

▷ Concepts being tested
▷ Relevant lectures
▷ Logical vs Graphical Objects
▷ Classes needed
LET’S DISCUSS CONTAINMENT DIAGRAMS
WHAT ARE Enums?
- A Java Enum is a special Java type used to define collections of constants.

WHY USE THEM?
- Correctness and Readability
- Increase compile-time checking and avoid errors from passing in invalid constants.

HOW IS IT USEFUL FOR SKETCHY?
- Enums will be useful in setting up your Radio Buttons.
To clarify the Enum help slide:

You should initialize your Enum in its own file, in this case the file would be called Direction.

```java
public enum Direction { NORTH, SOUTH, EAST, WEST; }
```

Then you can utilize your enums in any of your classes elsewhere in your program.

```java
Direction currDirection = Direction.SOUTH;

//elsewhere in the program:
switch(currDirection) {
    case NORTH:
        /*code*/
        case SOUTH:
            /*code*/
        /*code elided*/
}
```
How can we utilize **Inheritance** in Sketchy?

What are some different examples of commands that we will use in Sketchy?

How might undo() and redo() look different for fillColor() and resize()? What do we do when we have classes that might have similar methods but want to define those methods differently?

**Interface** vs **Abstract** Class?
INHERITANCE AND INTERFACES

Command can be abstract class or interface

Command

Raise Lower Rotate Resize Move Fill Delete CreateShape CreateLine

SketchyShape

SketchyRectangle Savable CurvedPolyLine

SketchyEllipse

EventHandler

All ButtonHandlers and MouseHandlers
Why is it important to have a **Layer Class**?

- A separate Layer class to keep track of the **three indices** needed (the ArrayList of **Saveables**, the ArrayList of **Sketchy Shapes**, and the **ObservableList of children**)

- **Sketchy Shapes** – A list of the Sketchy Shapes ordered from top to bottom that allows you to check if a shape is clicked on. Why is it important for the shapes to be in a top to bottom order?

- **Saveables** – A list of all the shapes and lines in a bottom to top order. Why is it important for the saveable objects to be in a bottom to top order?

- **ObservableList of children** (order observable in pane) – Keeps track of the index the shape’s node has in the Pane’s list of children
UNDO/REDO

Stack

Empty stack
Push
Push
Pop
What does `setFill` with undo/redo look like?

User presses Set Fill button with BLUE provided by ColorPicker (new color) and a RED SketchyEllipse selected on the canvas (old color)
SAVING & LOADING

- Important to consider Layering.
- Saving **shapes** vs **PolyLines**?
- File format + I/O
  - What information would you need to uniquely determine a curved line on the canvas pane?
  - What properties do you need to preserve?
SAVING & LOADING

NOTE

- FileIO support code provides:
  - `getFileName` will open save/load popup and return the String filename that the user types/selects
  - Takes in boolean that determines which window opens - true for save window, false for load window
  - You can use the String output from above to open a file for reading or writing
  - You can use `hasMoreData()` to check if there is anything else in the file
  - Always closeRead/Write after you are done
HIGH LEVEL DEMOS
resizeShape(Shape shape, Point2D prev, Point2D curr):
    rotation = shape.getRotation
    oldCenter = shape.getCenter
    rotatedPrev = rotatePoint(prev, oldCenter, rotation)
    rotatedCurr = rotatePoint(curr, oldCenter, rotation)

    dx = |rotatedCurr.getX - rotatedPrev.getX|
    dy = |rotatedCurr.getY - rotatedPrev.getY|

    if shrinking in the x direction:
        dx = -1 * dx
    if shrinking in the y direction:
        dy = -1 *dy

    set shape width to shape.getWidth + 2*dx
    set shape height to shape.getHeight + 2*dy

    // check if Shape center has changed and reset it if so
    newCenter = shape.getCenter
    if newCenter doesn’t equal oldCenter:
        dx = oldCenter.getX - newCenter.getX
        dy = oldCenter.getY - newCenter.getY
        set shape.x to shape.getX + dx
        set shape.y to shape.getY + dy
QUESTIONS?